BAB 5 IMPLEMENTASI

5.1 Lingkungan Pengujian

Penerapan *Support Vector Machine* (SVM) dalam klasifikasi susu sapi menggunakan dataset yang didapatkan dari UPT Laboratorium Kesehatan Hewan Malang menggunakan komputer perangkat keras yang di dalamnya terdapat perangkat lunak dengan spesifikasi seperti pada Tabel 5.1 dan Tabel 5.2.

Tabel 5.1 Lingkungan perangkat keras

Nama	**Spesifikasi**
Manufacture	Asus
Model	A 43 SV
Processor	Intel(R) Core(TM) i5-4200M CPU @ 2.50GHz (4 CPUs), ~2.5GHz
RAM	4096MB

Tabel 5.2 Lingkungan perangkat lunak

Nama	**Spesifikasi**
OS	Windows 7 Ultimate 64-Bit
Versi OS	7 Ultimate, Build 14393
BIOS *Version/Date*	InsydeH20 Version 03.73.0679CN33WW (V1.15).
Bahasa Pemrograman	Java
Tools	NetBeans IDE 8.2
DBMS	.xls (Microsoft Exel)

5.2 Batasan-batasan Implementasi

Batasan implementasi sistem klasifikasi kualitas susu sapi adalah sebagai berikut:

1. Aplikasi dibangun menggunakan bahasa pemrograman Java dan di jalankan menggunakan NetBean IDE 8.2.

2. Aplikasi hanya melakukan klasifikasi kualitas susu sapi menggunakan data yang didapatkan dari UPT Laboratorium Kesehatan Hewan Malang.
3. Pembahasan difokuskan pada implementasi metode *Support Vector Machine* (SVM) dan mengetahui kinerja (akurasi) *Support Vector Machine* (SVM) dengan membandingkan kernel RBF dan kernel Linier jika di terapkan pada klasifikasi kualitas susu sapi mengghunakan Dataset UPT Laboratorium Kesehatan Hewan Malang.

5.3 Implementasi Algoritme

Implementasi metode *Support Vector Machine* (SVM) untuk klasifikasi kualitas susu sapi terdapat 12 Algoritme. Berikut akan dijelaskan masing-masing Algoritmenya.

5.3.1 Proses Perhitungan Algoritme Kernel RBF

Proses perhitungan kernel *RBF* merupakan proses awal dalam metode *Support Vector Machine* (SVM). Berikut adalah *source code* dari proses perhitunga *kernel RBF*.

Tabel 5.3 *Source Code* Proses Perhitungan Kernel Linier

```
1  public void KernelRBF() {
2          System.out.println("");
3          for (int i = 0; i < trainingsvm1.length; i++) {
4              for (int j = 0; j < trainingsvm1.length; j++) {
5                  KernelRBF[i][j] = Math.exp(-1 *
6  ((Math.pow(trainingsvm1[i][0] - trainingsvm1[j][0], 2)
7                          + Math.pow(trainingsvm1[i][1] -
8  trainingsvm1[j][1], 2)
9                          + Math.pow(trainingsvm1[i][2] -
10 trainingsvm1[j][2], 2)
11                         + Math.pow(trainingsvm1[i][3] -
12 trainingsvm1[j][3], 2)
13                         + Math.pow(trainingsvm1[i][4] -
14 trainingsvm1[j][4], 2)
15                         + Math.pow(trainingsvm1[i][5] -
16 trainingsvm1[j][5], 2)
17                         + Math.pow(trainingsvm1[i][6] -
18 trainingsvm1[j][6], 2)) / (2 * Math.pow(sigma, 2))));
19             }
           }
           System.out.println("Kernel Gaussian RBF");
```

```
20          for (int i = 0; i < trainingsvm1.length; i++) {//
21              for (int j = 0; j < trainingsvm1.length; j++) {
22                  System.out.print(KernelRBF[i][j] + "|");
23              }
24              System.out.println("");
25          }
```

Berikut adalah penjelasan dari *source code* Tabel 5.3.

1. Baris 1 inisilaisasi method *kernel* RBF.
2. Baris 3-4 merupakan kode program untuk perulangan untuk data latih ke-*i* sampai dengan data latih ke-*j* sepanjang banyaknya data.
3. Baris 5-16 merupakan program untuk menghitung nilai dari *kernel* RBF.
4. Baris 19-20 merupakan perulangan untuk nilai *kernel* RBF ke-*i* dan ke-*j.*
5. Baris 23 merupakan kode program untuk mencetak nilai dari *kernel* RBF.

5.3.2 Proses Perhitungan Algoritme *Kernel Linier*

Proses perhitungan kernel linier merupakan proses awal dalam metode *Support Vector Machine* (SVM). Berikut adalah *source code* dari proses perhitunga *kernel Linier*.

Tabel 5.4 *Source Code* Proses Perhitungan Kernel Linier

```
1   public void KernelLinier() {
2           System.out.println("");
3           for (int i = 0; i < trainingsvm1.length; i++) {
4               for (int j = 0; j < trainingsvm1.length; j++) {
5                   KernelLinier[i][j] = (trainingsvm1[i][0] *
6   trainingsvm1[j][0])
7                                       +(trainingsvm1[i][1]
    * trainingsvm1[j][1])
8                                       +(trainingsvm1[i][2]
9   * trainingsvm1[j][2])
10                                      +(trainingsvm1[i][3]
11  * trainingsvm1[j][3])
12                                      +(trainingsvm1[i][4]
13  * trainingsvm1[j][4])
14                                      +(trainingsvm1[i][5]
    * trainingsvm1[j][5])
15
```

```
16                          +(trainingsvm1[i][6]
17  * trainingsvm1[j][6]);
18              }
19          }
20          System.out.println("Kernel Linier");
21          for (int i = 0; i < trainingsvm1.length; i++) {//
22              for (int j = 0; j < trainingsvm1.length; j++) {
23                  System.out.print(KernelLinier[i][j] + "|");
24              }
25              System.out.println("");
            }
```

Berikut adalah penjelasan dari *source code* Tabel 5.4.

1. Baris 1 inisilaisasi method *kernel* Linier.
2. Baris 3-4 merupakan kode program untuk perulangan untuk data latih ke-*i* sampai dengan data latih ke-*j* sepanjang banyaknya data.
3. Baris 5-16 merupakan program untuk menghitung nilai dari *kernel* Linier.
4. Baris 19-20 merupakan perulangan untuk nilai *kernel* Linier ke-*i* dan ke-*j.*
5. Baris 23 merupakan kode program untuk mencetak nilai dari *kernel* Linier.

5.3.3 Proses Perhitungan Algoritme *Sequential Training* SVM

Proses perhitungan *sequential training* merupakan tahap kedua untuk mencari *sequential training.* Berikut *source code* dari proses perhitungan *sequential training* :

Tabel 5.5 *Source Code* Proses Perhitungan Sequential Training

```
1           a.Matrix*Hessian*();
2           for (int i = 0; i < *itermax*; i++) {
3               a.NilaiEi();
4               a.delta_i();
5               a.ai();
6           }
```

Berikut adalah penjelasan dari *source code* Tabel 5.5.

1. Baris 1 pemanggilan method dari matriks *Hessian.*
2. Baris 2 melakukan perulangan sebanyak intputan iterasi yang di*input*kan.
3. Baris 3 pemanggilan method nilai E_i
4. Baris 4 melakukan pemanggilan method delta alpha.
5. Baris 5 melakukan pemanggilan method alpha I yang telah di perbaharui.

5.3.4 Proses Perhitungan Algoritme *Matriks Hessian*

Proses perhitungan nilai *Matriks Hessian* merupakan tahap ketiga dalam metode *Support Vector Machine* untuk mencari nilai Error. Berikut *source code* perhitungan nilai *Matriks Hessian*.

Tabel 5.6 Perhitungan Nilai *Matriks HessianKernel RBF*

```
1   public void Matrixhessian() {
2           System.out.println("");
            for (int i = 0; i < trainingsvm1.length; i++) {
3               for (int j = 0; j < trainingsvm1.length; j++) {
                    matrikshessian[i][j] = trainingsvm1[i][7] *
4   trainingsvm1[j][7] * (KernelRBF[i][j] + Math.pow(lambda, 2));
5
                }
6           }
7           System.out.println("Matriks Hessian");
            for (int i = 0; i < matrikshessian.length; i++) {//
8               for (int j = 0; j < matrikshessian.length; j++) {
9                   System.out.print(matrikshessian[i][j] + "|");
                }
10              System.out.println("");
11          }
        }
12
```

Berikut adalah penjelasan dari *source code* Tabel 5.6.

1. Baris 1 merupakan deklarasi method matriks *Hessian.*
2. Baris 2-3 merupakan kode program untuk perulangan untuk data latih ke-*i* sampai dengan data latih ke-*j* sepanjang banyaknya data.
3. Baris ke 4-5 merupakan perhitungan dari matriks *Hessian.*
4. Baris 7-8 merupakan merupakan perulangan untuk nilai matriks *Hessian* ke-*i* ke-*i* dan ke-*j.*
5. Baris 9 merupakan kode program untuk mencetak nilai dari matriks *Hessian.*

Tabel 5.7 Perhitungan Nilai *Matriks HessianKernel Linier*

```
1   public void Matrixhessian() {
2           System.out.println("");
            for (int i = 0; i < trainingsvm1.length; i++) {
3               for (int j = 0; j < trainingsvm1.length; j++) {
4                   matrikshessian[i][j] = trainingsvm1[i][7] *
trainingsvm1[j][7] * (KernelLinier[i][j] + Math.pow(lambda, 2));
5
                }
6           }
7           System.out.println("Matriks Hessian");
            for (int i = 0; i < matrikshessian.length; i++) {//
8               for (int j = 0; j < matrikshessian.length; j++) {
9                   System.out.print(matrikshessian[i][j] + "|");
                }
10              System.out.println("");
11          }
        }
12
```

Berikut adalah penjelasan dari *source code* Tabel 5.7.

1. Baris 1 merupakan deklarasi method matriks *Hessian.*
2. Baris 2-3 merupakan kode program untuk perulangan untuk data latih ke-*i* sampai dengan data latih ke-*j* sepanjang banyaknya data.
3. Baris ke 4-5 merupakan perhitungan dari matriks *Hessian.*
4. Baris 7-8 merupakan merupakan perulangan untuk nilai matriks *Hessian* ke-*i* ke-*i* dan ke-*j.*
5. Baris 9 merupakan kode program untuk mencetak nilai dari matriks *Hessian.*

5.3.5 Proses Perhitungan Algoritme Nilai *Ei*

Proses perhitungan nilai *Ei* merupakan tahap ke empat dalam metode *Support Vector Machine* untuk mencari nilai *Ei*. Berikut *source code* perhitungan nilai *Ei*.

Tabel 5.8 Perhitungan Nilai *Ei*

```
1   public void NilaiEi() {
2           System.out.println("");
3           double temp = 0;
4           for (int i = 0; i < trainingsvm1.length; i++) {
5               for (int j = 0; j < trainingsvm1.length; j++)
6   {
7                   temp += ai[j] * matrikshessian[i][j];
8               }
9               NilaiEi[i] = temp;
10              temp = 0;
11          }
```

```
12
13          System.out.println("Nilai Ei");
14          for (int i = 0; i < NilaiEi.length; i++) {
15              System.out.println(NilaiEi[i]);
16          }
17
18      }
```

Berikut adalah penjelasan *source code* Tabel 5.8.

1. Baris 1 merupakan deklarasi method nilai *Ei.*
2. Baris 3-4 merupakan kode program untuk perulangan untuk data latih ke-i sampai dengan data latih ke-j sepanjang banyaknya data.
3. Baris ke 7-10 merupakan perhitungan dari nilai *Ei.*
4. Baris 12-14 merupakan merupakan perulangan untuk nilai *Ei* sepanjang datanya.
5. Baris 15 merupakan kode program untuk mencetak nilai dari *Ei.*

5.3.6 Proses Perhitungan Algoritme Nilai $\delta\alpha_i$

Proses perhitungan nilai $\delta\alpha_i$ merupakan tahap ke lima dalam metode *Support Vector Machine* untuk mencari nilai $\delta\alpha_i$. Berikut *source code* perhitungan nilai $\delta\alpha_i$.

Tabel 5.9 Perhitungan Nilai $\delta\alpha_i$

```
1       public void delta_i() {
2           System.out.println("");
3           for (int i = 0; i < trainingsvm1.length; i++) {
4               delta_i[i]  =  Math.min(Math.max(gamma  *  (1  -
5       NilaiEi[i]), -ai[i]), C - ai[i]);
6           }
7
8           System,out,println("Nilai delta");
9           for (int i = 0; i < delta_i,length; i++) {
10              System.out.println(delta_i[i]);
11
12          }
13      }
```

Berikut adalah penjelasan *source code* Tabel 5.9.

1. Baris 1 merupakan deklarasi method nilai delta *alpha.*
2. Baris 3 merupakan kode program untuk perulangan untuk data latih ke-*i* sepanjang banyaknya data.

3. Baris ke 4-6 merupakan perhitungan dari nilai delta *alpha.*
4. Baris 9 merupakan merupakan perulangan untuk nilai delta alpha sepanjang datanya.
5. Baris 15 merupakan kode program untuk mencetak nilai delta *alpha.*

5.3.7 Proses Perhitungan Algoritme Nilai α_i

Proses perhitungan nilai α_i merupakan tahap ke enam dalam metode *Support Vector Machine* untuk mencari nilai α_i. Berikut *source code* perhitungan nilai α_i.

Tabel 5.10 Perhitungan Nilai α_i

```
1   public void ai() {
2           System.out.println("");
3           double temp = 0;
4           for (int i = 0; i < trainingsvm1.length; i++) {
5               temp = ai[i] + delta_i[i];
6               ai[i] = temp;
7               temp = 0;
8
9           }
10
11          System.out.println("Nilai ai ");
12          for (int i = 0; i < ai.length; i++) {
13              System.out.println(ai[i]);
14          }
15      }
```

Berikut adalah penjelasan *source code* Tabel 5.10.

1. Baris 1 merupakan deklarasi method nilai α_i.
2. Baris 4 merupakan kode program untuk perulangan untuk data latih ke-*i* sepanjang banyaknya data.
3. Baris ke 5-9 merupakan perhitungan dari nilai α_i.
4. Baris 12-14 merupakan merupakan perulangan untuk nilai α_i sepanjang datanya.
5. Baris 15 merupakan kode program untuk mencetak nilai dari ai α_i.

5.3.8 Proses Perhitungan Algoritme Nilai wx^+ dan wx^-

Pada perhitungan nilai wx^+ dan wx^- dimana nilai wx^+ didapatkan dari nilai alpha yang paling tinggi pada kelas positif dan nilai wx^- didaptkan dari nilai alpha yang paling tinggi pada kelas negatif kemudian proses bobot negatif yaitu perhitungan menggunkan kernel dari data latih pada kelas negatif dengan alpha tertinggi pada kelas negtif dan perhitungan bobot positif oleh kernel dari data latih yang memiliki

alpha paling tinggi pada kelas positif. Berikut *source code* perhitungan nilai W^+ dan W^- :

Tabel 5.11 Perhitungan nilai wx$^+$ dan wx$^-$ *RBF*

```
1   public double Knegatifpositif() {
2           double kelaspositif[][] = new double[3][2];
3           double kelasnegatif[][] = new double[6][2];
4           int c = 0;
5           int d = 0;
6
7           for (int i = 0; i < trainingsvm1.length; i++) {
8               if (trainingsvm1[i][7] == 1,0) {
9                   kelaspositif[c][0] = i;
10                  kelaspositif[c][1] = ai[i];
11                  c++;
12              } else if (trainingsvm1[i][7] == -1) {
13                  kelasnegatif[d][0] = i;
14                  kelasnegatif[d][1] = ai[i];
15                  d++;
16              }
17          }
18          double maxPositif = kelaspositif[0][1];
19          double urutan = kelaspositif[0][0];
20          for (int i = 0; i < kelaspositif.length; i++) {
21              if (maxPositif < kelaspositif[i][1]) {
22                  maxPositif = kelaspositif[i][1];
23                  urutan = kelaspositif[i][0];
24              }
25
26          }
27          double maxNegatif = kelasnegatif[0][1];
28          double urutanNegatif = kelasnegatif[0][0];
29          for (int i = 0; i < kelasnegatif.length; i++) {
30              if (maxNegatif < kelasnegatif[i][1]) {
31                  maxNegatif = kelasnegatif[i][1];
32                  urutanNegatif = kelasnegatif[i][0];
33              }
34          }
35                  double sumA = 0.0;
36
37          for (int i = 0; i < trainingsvm1.length; i++) {
38              double hasil = (ai[i] * trainingsvm1[i][7] *
39  KernelRBF[i][(int) urutan]);
40              sumA += hasil;
41          }
42
43          double sumB = 0.0;
44          for (int i = 0; i < trainingsvm1.length; i++) {
45              double hasil = (ai[i] * trainingsvm1[i][7] *
46  KernelRBF[i][(int) urutanNegatif]);
47              sumB += hasil;
48          }
49          System.out.println("Nilai Kx+ = " + urutan);
```

```
50          System.out.println("Nilai Kx- = " +
51      urutanNegatif);
52          System.out.println("Kernel(xi,x+)    = " + sumA);
53          System.out.println("Kernel(xi,x-)   = " + sumB);
```

Berikut adalah penjelasan *source code* Tabel 5.11.

1. Baris 1 merupakan deklarasi method kelas positif negatif
2. Baris 38-40 merupakan perulangan untuk menghitung bobot dari data latih yang memiliki kelas level positif (+1)
3. Baris 44-47 merupakan perulangan untuk menghitung bobot dari data latih yang memiliki kelas level negatif (-1)

Tabel 5.12 Perhitungan nilai wx^+ dan wx^- *Linier*

```
1       public double Knegatifpositif() {
2           double kelaspositif[][] = new double[3][2];
3           double kelasnegatif[][] = new double[6][2];
4           int c = 0;
5           int d = 0;
6
7           for (int i = 0; i < trainingsvm1.length; i++) {
8               if (trainingsvm1[i][7] == 1,0) {
9                   kelaspositif[c][0] = i;
10                  kelaspositif[c][1] = ai[i];
11                  c++;
12              } else if (trainingsvm1[i][7] == -1) {
13                  kelasnegatif[d][0] = i;
14                  kelasnegatif[d][1] = ai[i];
15                  d++;
16              }
17          }
18          double maxPositif = kelaspositif[0][1];
19          double urutan = kelaspositif[0][0];
20          for (int i = 0; i < kelaspositif.length; i++) {
21              if (maxPositif < kelaspositif[i][1]) {
22                  maxPositif = kelaspositif[i][1];
23                  urutan = kelaspositif[i][0];
24              }
25
26          }
27          double maxNegatif = kelasnegatif[0][1];
28          double urutanNegatif = kelasnegatif[0][0];
29          for (int i = 0; i < kelasnegatif.length; i++) {
30              if (maxNegatif < kelasnegatif[i][1]) {
31                  maxNegatif = kelasnegatif[i][1];
32                  urutanNegatif = kelasnegatif[i][0];
33              }
```

```
34          }
35                  double sumA = 0.0;
36
37          for (int i = 0; i < trainingsvm1.length; i++) {
38              double hasil = (ai[i] * trainingsvm1[i][7] *
39  KernelLinier[i][(int) urutan]);
40              sumA += hasil;
41          }
42
43          double sumB = 0.0;
44          for (int i = 0; i < trainingsvm1.length; i++) {
45              double hasil = (ai[i] * trainingsvm1[i][7] *
46  KernelLinier[i][(int) urutanNegatif]);
47              sumB += hasil;
48          }
49          System.out.println("Nilai Kx+ = " + urutan);
50          System.out.println("Nilai Kx- = " +
51  urutanNegatif);
52          System.out.println("Kernel(xi,x+)   = " + sumA);
53          System.out.println("Kernel(xi,x-)  = " + sumB);
```

Berikut adalah penjelasan *source code* Tabel 5.12.

1. Baris 1 merupakan deklarasi method kelaspositifnegatif
2. Baris 38-40 merupakan perulangan untuk menghitung bobot dari data latih yang memiliki kelas level positif (+1)
3. Baris 44-47 merupakan perulangan untuk menghitung bobot dari data latih yang memiliki kelas level negatif (-1)

5.3.9 Proses Perhitungan Algoritme Nilai Bias

Proses perhitungan berikutnya adalah untuk mencari nilai bias. Berikut *source code* untuk mencari nilai bias.

Tabel 5.13 Perhitungan Nilai Bias

```
1
2           bias = -0.5 * (sumA + sumB);
3
4           System.out.println("bias = " + bias);
5           return bias;
6       }
```

5.3.10 Proses Algoritme Perhitungan Nilai *Kernel Test*

Pada perhitungan nilai *kernel test* ini menggunakan data uji untuk menghitung *kernel*. Berikut source code untuk mencari nilai *kernel test*.

Tabel 5.14 Perhitungan Nilai *Kernel Test RBF*

```
1       public void tkernelRBF() {
2            System.out.println("");
3            for (int i = 0; i < testing.length; i++) {
4                for (int j = 0; j < trainingsvm1.length;) {
5                    hasil         =         Math.exp(-1         *
6       ((Math.pow(testing[i][0] - trainingsvm1[j][0], 2)
7                               +      Math.pow(testing[i][1]      -
8       trainingsvm1[j][1], 2)
9                               +      Math.pow(testing[i][2]      -
10      trainingsvm1[j][2], 2)
11                              +      Math.pow(testing[i][3]      -
12      trainingsvm1[j][3], 2)
13                              +      Math.pow(testing[i][4]      -
14      trainingsvm1[j][4], 2)
15                              +      Math.pow(testing[i][5]      -
16      trainingsvm1[j][5], 2)
17                              +      Math.pow(testing[i][6]      -
18      trainingsvm1[j][6], 2)) / (2 * Math.pow(sigma, 2))));
19                   tkernelRBF[i][j] = hasil;
20                   j++;
21               }
22           }
23
24           System.out.println("tkernel");
25           for (int j = 0; j < testing.length; j++) {
26               for (int i = 0; i < trainingsvm1.length; i++) {//
27                   System.out.println(tkernelRBF[j][i] + "\t");
28               }
29               System.out.println("");
30           }
31
32           System.out.println("");
33
34
```

Berikut adalah penjelasan *source code* Tabel 5.14 :

1. Baris 1, inisilaisasi method testing *kernel* RBF.
2. Baris 3-4 merupakan kode program untuk perulangan untuk data testing ke-*i* sampai dengan data latih ke-*j* sepanjang banyaknya data.
3. Baris 5-23 merupakan program untuk menghitung nilai dari testing *kernel* RBF.
4. Baris 25-26 merupakan perulangan untuk nilai *kernel* RBF ke-*i* dan ke-*j.*
5. Baris 28 merupakan kode program untuk mencetak nilai dari *kernel* RBF.

Tabel 5.15 Perhitungan Nilai *Kernel Test Linier*

```
1       public void tkernelRBF() {
2            System.out.println("");
3            for (int i = 0; i < testing.length; i++) {
4                for (int j = 0; j < trainingsvm1.length;) {
5                    hasil = (testing[i][0] * trainingsvm1[j][0])
6                            +          (testing[i][1]          *
7       trainingsvm1[j][1])
8                            +          (testing[i][2]          *
9       trainingsvm1[j][2])
10                           +          (testing[i][3]          *
11      trainingsvm1[j][3])
12                           +          (testing[i][4]          *
13      trainingsvm1[j][4])
14                           +          (testing[i][5]          *
15      trainingsvm1[j][5])
16                           +          (testing[i][6]          *
17      trainingsvm1[j][6]);
18                   tkernelLinier[i][j] = hasil;
19                   j++;
20               }
21           }
22
23           System.out.println("tkernel");
24           for (int j = 0; j < testing.length; j++) {
25               for (int i = 0; i < trainingsvm1.length; i++) {//
26                   System.out.println(tkernelLinier[j][i]     +
27      "\t");
28               }
29               System.out.println("");
30           }
31
32           System.out.println("");
33
34
```

Berikut adalah penjelasan *source code* Tabel 5.15.

1. Baris 1, inisilaisasi method testing *kernel* Linier.
2. Baris 3-4 merupakan kode program untuk perulangan untuk data testing ke-*i* sampai dengan data latih ke-*j* sepanjang banyaknya data.
3. Baris 5-23 merupakan program untuk menghitung nilai dari testing *kernel* Linier.
4. Baris 25-26 merupakan perulangan untuk nilai *kernel* Linier ke-*i* dan ke-*j.*
5. Baris 28 merupakan kode program untuk mencetak nilai dari *kernel* Linier.

5.3.11 Proses Perhitungan Algoritme Nilai *f(x)*

Proses selanjutnya adalah menentukan nilai *f(x)*. Berikut *source code* perhitungan nilai *f(x)* :

Tabel 5.16 Perhitungan Nilai *f(x)*

```
1       System.out.println("Nilai aiyk");
2       for (int j = 0; j < trainingsvm2.length; j++) {
3       for (int k = 0; k <
4       ai2.length; k++) {total2 = ai2[k] * trainingsvm2[j][7] *
5       tkernelRBF2[i][j];
6
7                           }
8                           System.out.println(total2);
9                           total4 += total2;
10                      }
11                      System.out.println("total");
12                      System.out.println(total4);
13                      System.out.println("");
14                      fx2 = (total4 + bias2);
15                      System.out.println("nilai fx");
16                      System.out.println(fx2);
17                      total2 = 0;
18                      total4 = 0;
19                      if (fx2 >= 0) {
20                          hasilUji[i][1] = fx2;
21                          hasilUji[i][0] = 2;
22
23                      } else {
24                          hasilUji[i][1] = fx2;
25                          hasilUji[i][0] = 3;
26                      }
27                  }
                }
```

Berikut adalah penjelasan *source code* Tabel 5.16.

1. Baris 2-3 merupakan perulangan untuk nilai dari $\alpha_i y_i K$.
2. Baris 4-5 perhitungan nilai $\alpha_i y_i K$.
3. Baris 7-10 perhitungan nilai Fx level 1.
4. Baris 11-14 perhitungan nilai Fx level 2.
5. Baris 19-22 Kondisi jika nilai Fx lebih dari 0 maka akan masuk ke kelas 2.
6. Baris 24-26 Kondisi jika nilai Fx kurang dari 0 maka akan masuk ke kelas 3.

5.4 Hasil Implementasi

Pada bab ini merupakan hasil tampilan program. Tampilan program ini meliputi *dataset*, perhitungan *matriks kernel*, perhitungan *matriks hessian* dan hasil klasifikasi.

5.4.1 Tampilan *Dataset*

DATA TRAINING SVM LEVEL 1

3.33	10.36	37.35	3.81	5.7	0.0	20.0	1.0
5.09	9.32	31.22	3.44	5.12	0.0	11.0	1.0
5.11	9.27	30.99	3.61	5.09	0.0	32.0	1.0
4.39	9.91	27.91	3.24	5.01	0.0	29.0	1.0
3.15	7.08	30.99	2.87	4.23	0.0	22.0	1.0
4.49	7.74	24.15	2.86	4.25	0.0	24.0	1.0
3.03	8.17	26.49	2.74	4.57	1.8	24.0	1.0
2.8	7.29	28.76	2.7	4.09	2.1	30.0	1.0
2.28	6.84	25.66	2.79	4.46	0.0	27.0	1.0
3.14	8.13	22.52	2.84	4.33	13.0	24.0	1.0
1.01	6.29	23.03	2.31	3.46	25.8	18.0	-1.0
2.69	7.54	11.3	1.3	1.94	99.9	32.0	-1.0
1.53	5.01	18.56	2.21	3.1	29.0	18.0	-1.0
1.51	3.22	17.04	2.18	1.76	9.9	18.0	-1.0
1.61	3.79	12.96	1.39	3.08	20.0	24.0	-1.0

DATA TRAINING SVM LEVEL 2

3.33	10.36	37.35	3.81	5.7	0.0	20.0	1.0
5.09	9.32	31.22	3.44	5.12	0.0	11.0	1.0
5.11	9.27	30.99	3.61	5.09	0.0	32.0	1.0
4.39	9.91	27.91	3.24	5.01	0.0	29.0	1.0
3.15	7.08	30.99	2.87	4.23	0.0	22.0	1.0
4.49	7.74	24.15	2.86	4.25	0.0	24.0	-1.0
3.03	8.17	26.49	2.74	4.57	1.8	24.0	-1.0
2.8	7.29	28.76	2.7	4.09	2.1	30.0	-1.0
2.28	6.84	25.66	2.79	4.46	0.0	27.0	-1.0
3.14	8.13	22.52	2.84	4.33	13.0	24.0	-1.0
1.01	6.29	23.03	2.31	3.46	25.8	18.0	-1.0
2.69	7.54	11.3	1.3	1.94	99.9	32.0	-1.0
1.53	5.01	18.56	2.21	3.1	29.0	18.0	-1.0
1.51	3.22	17.04	2.18	1.76	9.9	18.0	-1.0
1.61	3.79	12.96	1.39	3.08	20.0	24.0	-1.0

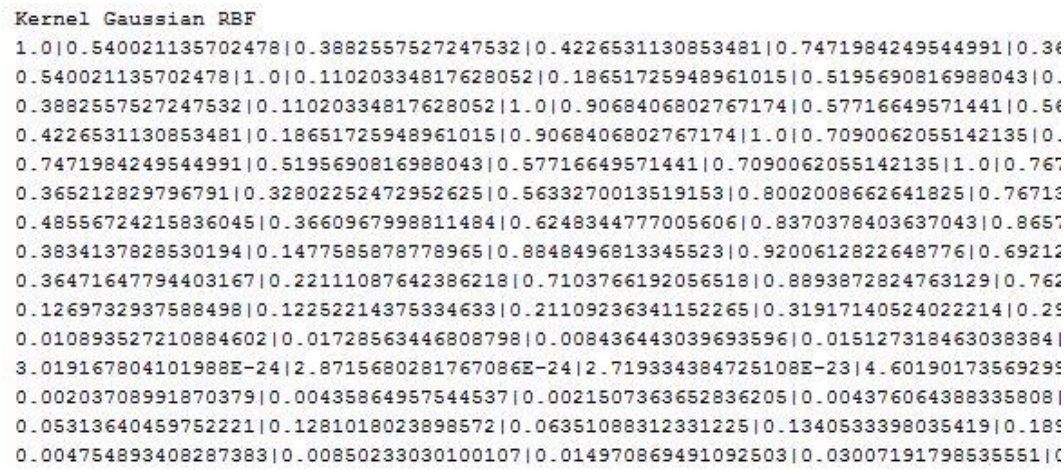
DATA TESTING

2.14	6.42	22.32	2.23	3.55	14.0	23.0	1.0
3.03	5.58	18.76	2.06	3.16	20.0	14.0	1.0
2.56	7.95	30.14	2.87	4.01	0.0	28.0	2.0
3.5	6.8	26.68	3.01	4.11	11.0	31.0	2.0
3.7	8.81	29.92	3.2	5.3	2.0	19.0	3.0
2.56	9.87	28.87	3.27	4.45	0.0	13.0	3.0

Gambar 5.1 Tampilan *Dataset*

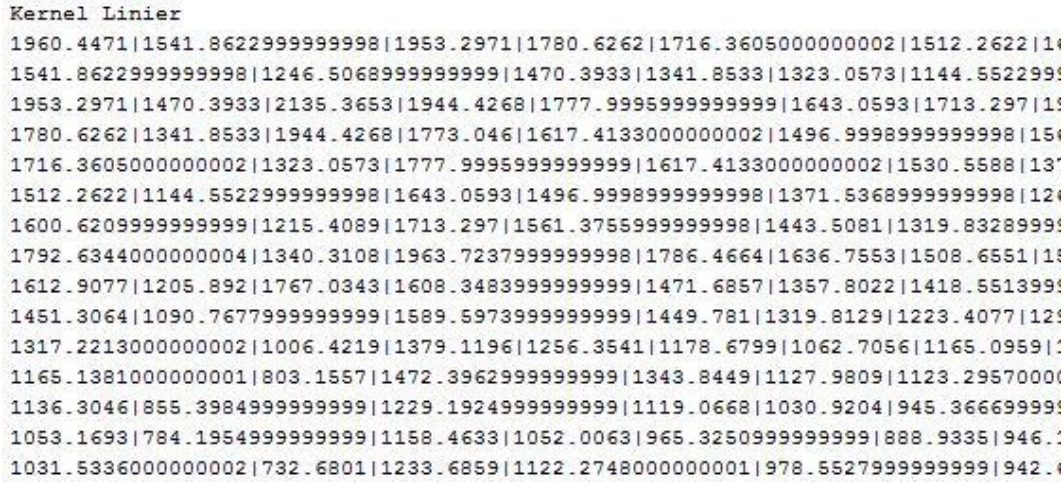
Pada Gambar 5.1 merupakan tampilan dataset training level 1, level 2 dan testing.

5.4.2 Tampilan *Matriks Kernel*



Gambar 5.2 Tampilan Hasil *Matriks RBF*

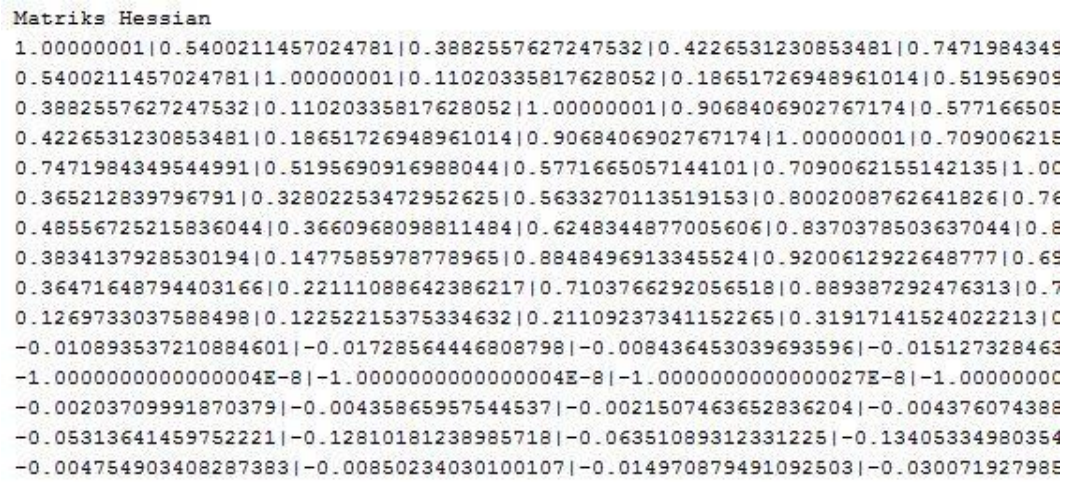
Pada Gambar 5.2 merupakan tampilan hasil perhitungan *matriks kernel RBF*.



Gambar 5.3 Tampilan Hasil *Matriks Linier*

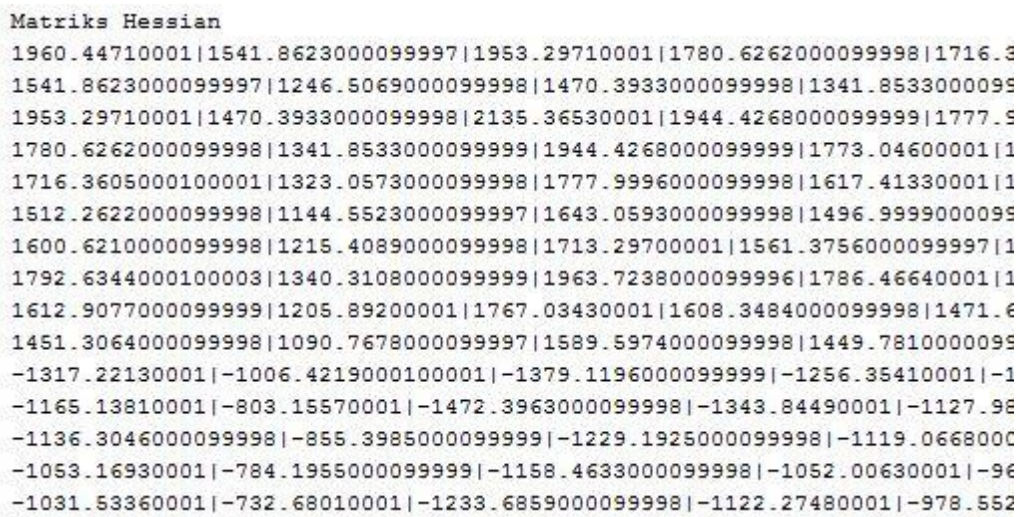
Pada Gambar 5.3 merupakan tampilan hasil perhitungan *matriks kernel Linier*.

5.4.3 Tampilan *Matriks Hessian*



Gambar 5.4 Tampilan *Matriks Hessian RBF*

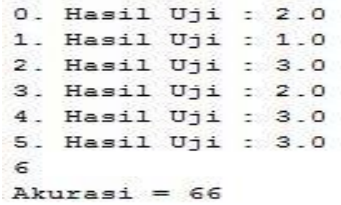
Pada Gambar 5.4 merupakan tampilan hasil *matriks hessian kernel RBF*.



Gambar 5.5 Tampilan *Matriks Hessian Linier*

Pada Gambar 5.5 merupakan tampilan hasil *matriks hessian kernel Linier*.

5.4.4 Tampilan Hasil Klasifikasi



Gambar 5.6 Tampilan akurasi dan hasil klasifikasi